Cupcake

Navn: Mariam Elmir, Asger Junker, Daniel Jacobsen, Rasmus Svendsen

Github-navn: MariamE04, Asger191, DanielBasco, RasmusVJS

CPH-Mails: cph-me344@cphbusiness.dk , cph-aj599@cphbusiness.dk, cph-dj205@cphbusiness.dk, cph-rs264@cphbusiness.dk

Gruppe: Gruppe 16

Klasse: Hold A

Dato: 24-03-2025

Indholdsfortegnelse

Indledning

Denne her rapport dokumenterer udviklingen af et bestillingssystem for Olskers Cupcakes, som er en bornholmsk cupcakevirksomhed. Opgaven går ud på at designe et system, som skal være brugervenligt, hvor en kunde kan bestille og betale for deres cupcake, samt administrere deres ordre.

Projektet tager udgangspunkt i en række user stories, som beskriver de funktionelle krav for både kunder og administratoren. Vores mål er at udvikle en første prototype ved at fokusere på kernefunktionaliteterne, før vi udvider systemet med flere features.

Denne rapport er skrevet til en fagfælle. Med andre ord en anden datamatikerstuderende på 2. semester, der har den tekniske forståelse, men som ikke har kendskab til Olskers Cupcakes-projekt på forhånd. Derfor vil rapporten fokusere på at forklare de vigtigste aspekter af projektet, herunder systemets arkitektur, funktionalitet og implementeringsvalg.

Udover de funktionelle krav stilles der også tekniske krav til systemet, herunder brug af Java 17, Javalin og en Postgres-database, der skal normaliseres til tredje normalform så vidt muligt. Desuden skal kildekoden hostes på GitHub, og systemet skal være responsivt, så det kan fungere både på laptops og mobiltelefoner. Disse tekniske krav er opfyldte, udover at vi i gruppen har besluttet at bruge Java 23.

Baggrund

Olsker Cupcakes er en bornholmsk cupcakevirksomhed som fokuserer på dybdeøkologi og bæredygtighed. Virksomheden ønsker at modernisere deres bestillingsproces ved at udvikle et digitalt system, som gør det nemmere for kunder at bestille og betale for cupcakes online.

Systemet skal være brugervenligt, både for at kunne imødekomme kundernes såvel som administratorens behov. Kunderne skal være i stand til at sammensætte deres egen cupcake, ved at vælge en bund og en top, som kan lægges i en indkøbskurv, så deres ordre kan administreres.

En særlig detalje for dette system er, at betaling ikke fungerer på samme måde som i en traditionel webshop. I stedet indsætter administratoren et beløb direkte ind i databasen på kundens konto, og når kunden foretager et køb, trækkes beløbet fra kontoen. Denne betalingsform kræver en backend-løsning, hvor administratoren har adgang til at opdatere databasen (Postgres) og hvor kundernes betalinger registreres korrekt.

Administratoren skal kunne have et overblik over indkomne ordrer og kundedata, samt have mulighed for at administrere systemets indhold. For at sikre en effektiv drift skal systemet integreres med en database, hvor ordrer og kundeoplysninger lagres.

Og som nævnt i introen er det givet, at projektet er baseret på en række user stories, der beskriver de funktionelle krav til systemet.

Teknologivalg

Projektet benytter følgende teknologier og versioner:

- Java 23
- Javalin 6.5.0
- Thymeleaf 3.1.3.RELEASE
- PostgreSQL JDBC Driver 42.7.5
- HikariCP 6.2.1
- IntelliJ IDEA 2023.3.5

Disse teknologier blev valgt for at understøtte udviklingen af en webapplikation, som håndterer databaser, webserver-funktionalitet og skabelonrendering effektivt.

Krav

Via dette system håber virksomheden at kunne levere et effektivt og brugervenligt bestillingssystem, som giver kunderne mulighed for nemt at bestille og betale for cupcakes og dermed optimere kundeoplevelsen og ledelsesprocessen. Systemet skal også gøre det muligt for administratorer at behandle ordrer og kunder effektivt og samtidig understøtte virksomhedens fokus på bæredygtighed og dyb økologi.

User Stories

US-1: Som kunde kan jeg bestille og betale cupcakes med en valgfri bund og top, sådan at jeg senere kan køre forbi butikken i Olsker og hente min ordre.

US-2 Som kunde kan jeg oprette en konto/profil for at kunne betale og gemme en en ordre.

US-3: Som administrator kan jeg indsætte beløb på en kundes konto direkte i Postgres, så en kunde kan betale for sine ordrer.

US-4: Som kunde kan jeg se mine valgte ordrelinier i en indkøbskurv, så jeg kan se den samlede pris.

US-5: Som kunde eller administrator kan jeg logge på systemet med email og kodeord. Når jeg er logget på, skal jeg kunne se min email på hver side (evt. i topmenuen, som vist på mockup'en).

US-6: Som administrator kan jeg se alle ordrer i systemet, så jeg kan se hvad der er blevet bestilt.

US-7: Som administrator kan jeg se alle kunder i systemet og deres ordrer, sådan at jeg kan følge op på ordrer og holde styr på mine kunder.

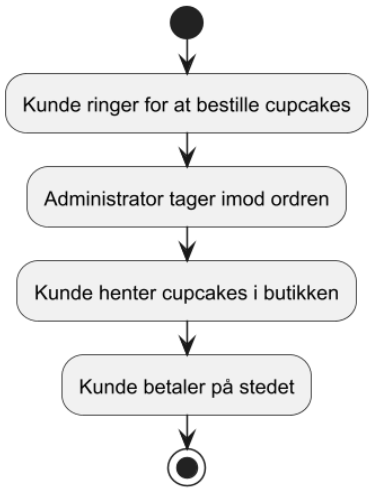
US-8: Som kunde kan jeg fjerne en ordrelinie fra min indkøbskurv, så jeg kan justere min ordre.

US-9: Som administrator kan jeg fjerne en ordre, så systemet ikke kommer til at indeholde udgyldige ordrer. F.eks. hvis kunden aldrig har betalt.

Aktivitetsdiagram

For at få overblik over bestillingsprocessen i Olskers Cupcakes har vi i gruppen udviklet et aktivitetsdiagram, som skal beskrive workflowet før og efter digitalisering. Det hjælper nemlig os med at forstå, hvilke forretningsprocesser vi skal understøtte i systemet. Uderover det så hjalp disse diagrammer os med at identificere nødvendige funktioner, optimere processen og sikre, at vi opfyldte de funktionelle krav fra vores user stories.

AS-IS: Nuværende workflow

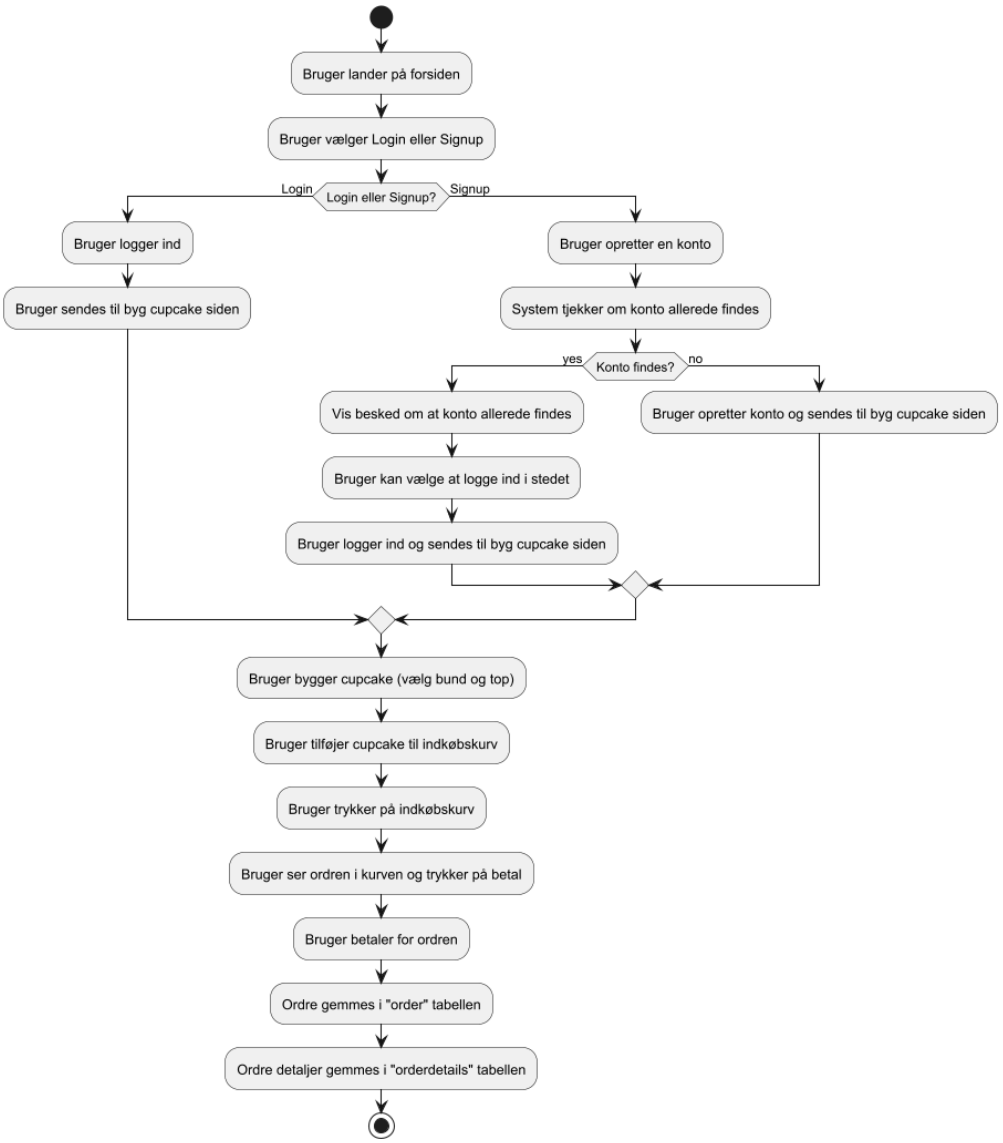


I Gruppen startet vi med at analyser den eksisterende bestillingsproces, som resulteret i at vi lavet en AS-IS aktivitetsdiagram. Diagrammet er lavet på baggrund af research og dialog om, hvordan ordre bliver håndteret i dag. Det vi så kom frem til er, at hele processen foregår manuelt:

- Kunden ringer for at bestille cupcakes
- Administratoren noterer ordren.
- Kunden afhenter og betaler i butikken

Dette identificeret at der er nogle udfordringer, såsom begrænset fleksibilitet for kunder og at processen kræver manuel håndtering af bestillinger og betalinger, hvilket kan føre til fejl og ekstra arbejde for personalet.

TO-BE: Fremtidigt workflow



Baseret på de funktionelle krav altså user stories udviklede vi et TO-BE aktivitetsdiagram, der viser den digitale løsning. Vi valgte at lægge fokuset på de vigtigste funktioner fra user stories herunder:

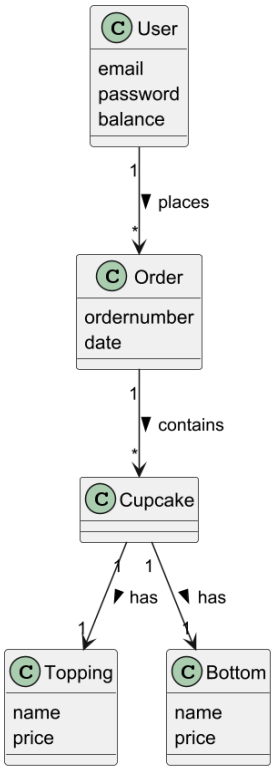
- US-1 & US-4: Kunden kan designe sin cupcake og tilføje den til indkøbskurven.
- US-2 & US-5: Kunden kan oprette en konto og logge ind.
- US-3: Betaling håndteres via kontosystemet i Postgres.
- US-6: Administratoren kan håndtere ordrer og kunder i systemet.

Diagrammet viser en mere automatiseret proces, hvor kunden har mulighed for selv at stå for bestilling via hjemmesiden, og data lagres direkte i systemets database (Postgres). Dette reducerer nemlig den manuelle arbejdsbyrde og giver kunden større fleksibilitet.

Ved at sammenligne begge diagrammer AS-IS og TO-BE kunne vi identificere vigtige ændringer og sikre, at vores system løser disse problemer, den nuværende proces har. Diagrammerne fungerede også som hjælp til at strukturere systemets arkitektur, herunder databaseopsætning.

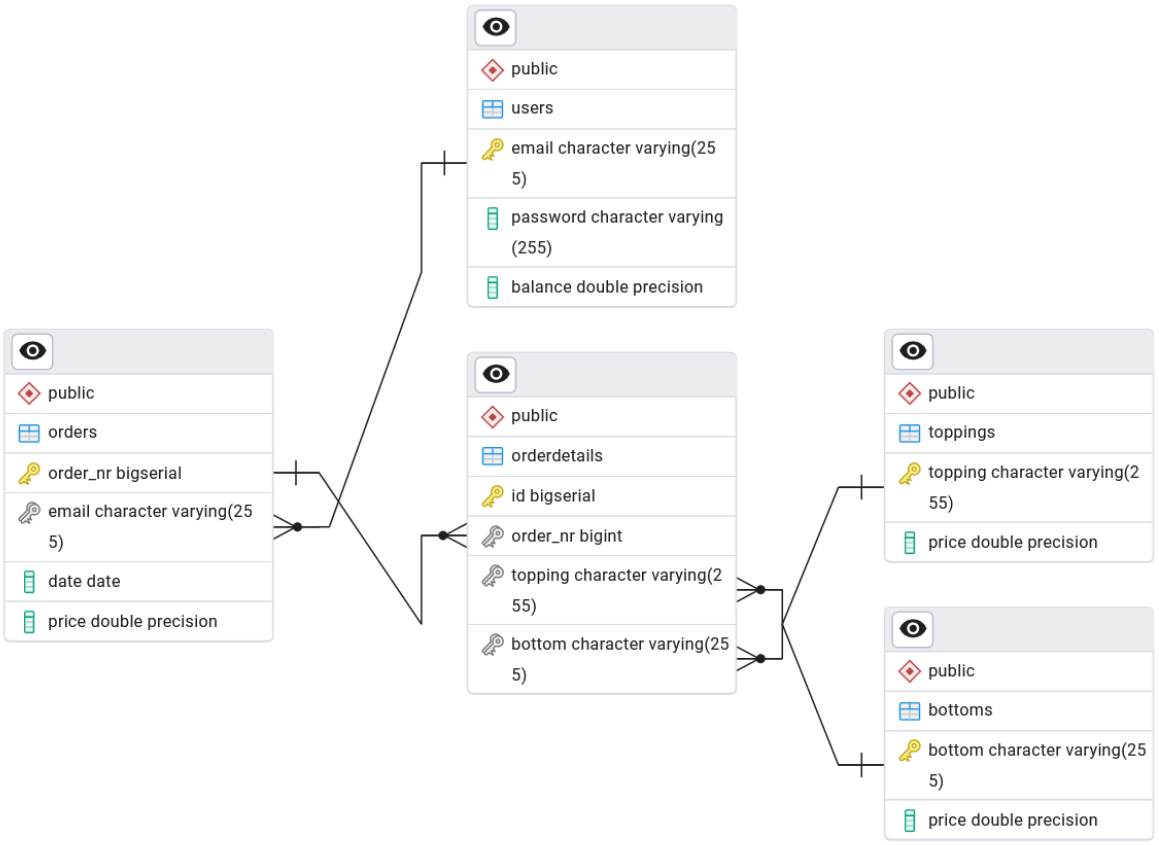
Domænemodel og ERDiagram

Domænemodel



For at opnå de satte krav, blev det anslået, at disse fem entitetstyper var nødvendige i systemet. Der skal være brugerkontoer, med en opgørelse, som administratoren kan manipulere, og med unikt login, opnået via mail og password. Disse brugere skal dertil være i stand til at oprette ordrer, der fortæller administratoren, hvilke cupcakes, brugeren har bestilt, og hvad den samlede pris på disse er. Hver cupcake består yderligere af sammensætningen af én top og én bund, hvorfra den individuelle cupcakes pris udregnes.

ERDiagram



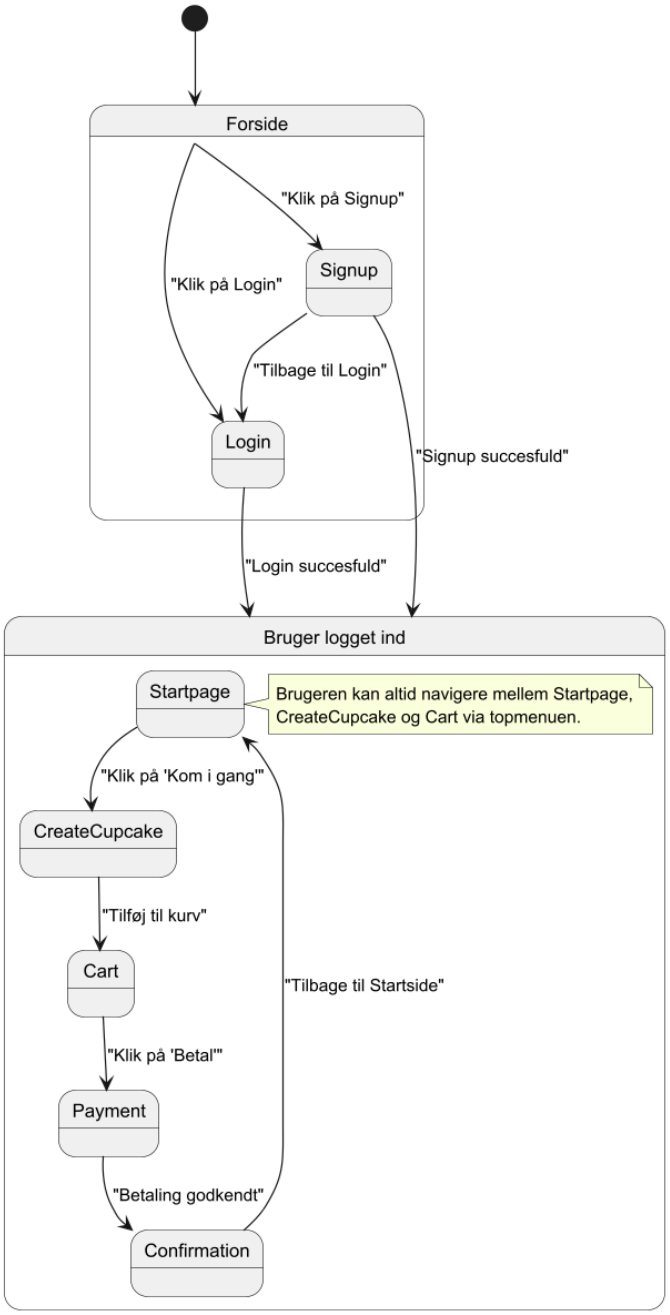
I dette ERD ses, hvordan domænemodellen udvikles i databasen. Hertil ses det, at selvom Cupcake er en entitet i domænemodellen, eksisterer den ikke direkte i databasen. I stedet, sættes top og bund sammen i orderdetailstabellen. Den betydningsfulde forskel mellem orderdetails og cupcakes er, at en orderdetail er en underordrer, der oprettes, for hvert individuelt cupcake køb, mens cupcakes er objektet, der købes. Da cupcakes ikke har nogen indædte data, har den derfor ingen relevans til databasen. I stedet indeholder orderdetailstabellen en record fra toppingstabellen og en fra bottomstabellen, som tilsammen repræsenterer den bestilte cupcake.

Både orders- og orderdetailstabellen har en integersekvens som primærnøgle, order_nr og id, da de ikke har nogen indædt kombination af data, som ikke kunne eksistere i en anden record. Toppings- og bottomstabellen har strengen name som primærnøgle, da det aldrig burde ske, at to forskellige toppe eller bunde havde det samme navn, så de kan unikt identificeres gennem deres navn. Usertabellen har strengen email som primærnøgle, da alle brugere skal have en emailadresse, og denne skal også være unik for den specifikke bruger. Deraf begrundes det, at email er en tilstrækkelig primærnøgle. Tabellernes primærnøgler er derudover også de fremmednøgler, der bruges til at forbinde tabellerne.

Vedrørende normalisering, overholder tabellerne nærmest at være på 3. normalform. Dog, har orderstabellen en kolonne, der indeholder ordrens pris, på baggrund af, at hvis cupcakepriserne ændres over tid, vil administratoren stadig være i stand til at se, hvad der blev betalt for en tidligere ordre. Dette er også grunden til, at prisernes datatype er en double, i stedet for en integer, som ellers ville være dækkende for de nuværende priser.

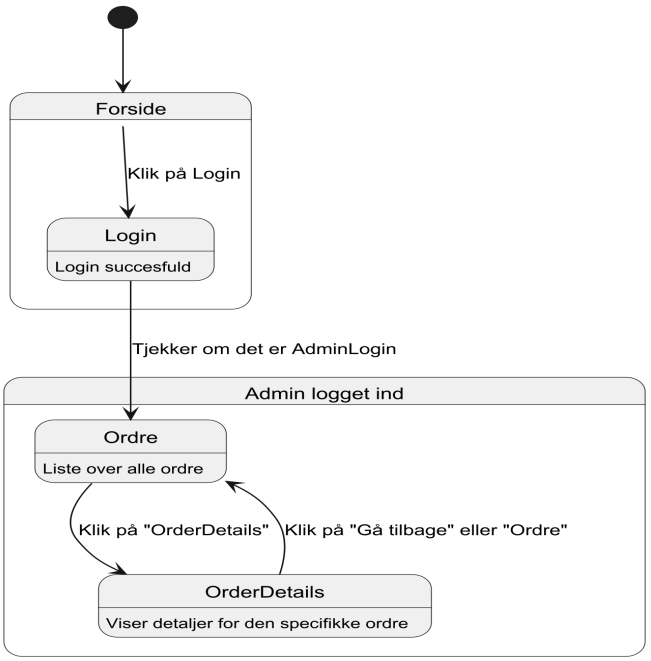
Da ordrens pris er bestemt af de tilhørende orderdetails toppe og bunde, samt hvad priserne var den dato, og derfor ikke er direkte knyttet til ordren selv, overholder dette ikke 3. normalform. En måde, hvorpå 3. normalform kunne overholdes, hvor denne information stadig kunne være tilgængelig, ville være, hvis der var en tabel over, hvilke priser de forskellige toppe og bunde havde gennem tiden. Men, da dette ville være en masse arbejde og data, for en funktion af lav prioritet, er dette ikke gjort, og lidt mindre normaliseret data er anset som en acceptabel konsekvens.

Navigationsdiagram



Når man kommer ind på vores website, starter man med en lille beskrivelse af Olsker Cupcakes, og en valgmulighed mellem at logge ind eller signup. Når man har logget ind eller trykket på “SignUp” bliver man automatisk smidt ind på vores forside, hvor vi har en navigationsbar med funktionerne “Hjem”, “Byg CupCake”, “Email” og ens indkøbskurv. Disse elementer er synlige for kunden uanset hvorhenne på hjemmesiden vedkommende er. “Hjem” knappen gør, at man kan komme

tilbage til startsiden nemt og hurtigt, hvor man har muligheden, for at trykke på en knap der hedder "Kom igang", hvor man bliver renderet hen til siden, hvor man kan bygge sin egen cupcake og lægge den i sin indkøbskurv. "Byg CupCake" har samme funktion som "Kom igang" knappen på startsiden, men den kan man som sagt tilgå hvor end man er på hjemmesiden via. navigationsbaren. Ideen med navigationsbaren er at den skal være synlig og ikke til at misforstå. Det skal gøres så nemt og enkelt for kunden som overhovedet muligt, at navigere rundt på hjemmesiden og bygge, bestille, gå tilbage eller hvad end kundens formål på hjemmesiden er.



(Navigationsdiagram for Admin)

Hvis man har et admin login, bliver man renderet en "admin" side, hvor man kan se en liste af alle ordrer der er registreret i databasen. Layoutet for ordren er ordre nummer, dato og kundens email. Ud fra ordre dataen kan man trykke på en knap der hedder "Order Details". Når denne knap trykkes på bliver admin renderet hen til en underside, hvor man kan tilgå detaljerne for den specifikke ordre. Detaljerne for ordren består af ordre nummer, bunden og toppen for den selvbyggede cupcake. Hvis admin ønsker at komme tilbage til ordre siden, så har admin både muligheden for at trykke på "ordre" i navigationsbaren eller knappen "gå tilbage" som også befinder sig i navigationsbaren. Siden for den eksklusive oprettede bruger også kaldet admin, har en

mere simpel navigationsbar, hvor der kun står email og “Ordre”, som renderer en hen til listen af ordre.

Særlige forhold

I det her afsnit beskrives de særlige forhold, der er implementeret i programmet for at sikre en stabil og sikker applikation.

Session-håndtering

I programmet benyttes sessioner til at gemme brugerinformation, så brugeren forbliver logget ind under en session. Dette kan gøres ved at lagre brugeroplysninger i session via ctx.sessionAttribute("currentUser", user); i HomeController. Uderover det håndteres administratorkontrol ved at gemme en session variable med ctx.sessionAttribute("admin", loggedInUser);, hvilket gøre det muligt at have forskellig adgang afhængigt af brugens rolle.

For at sikre en bedre brugeroplevelse gemmes der også indkøbskurven i sessionen. Dette sker i CupCakeController klassen via ctx.sessionAttribute("cart", cart);, så indholdet af kurven bevares, mens brugeren navigerer gennem systemet.

En ekstra session vi har lavet håndtere ordreoplysninger, denne funktionalitet er dog forbeholdt til administratorer. I getOrderDetailsByOrderNumber(Context ctx) gemmes ordreoplysningerne i sessionen ctx.sessionAttribute("orderDetails", orderDetails);. Dette gøre det muligt at vise ordredata i HTML-siden rderdetails.html, hvor Thymeleaf bruges til at hente data direkte fra sessionen.

Exception-håndtering

Der benyttes en specifik exception-type, DatabaseException til at håndtere fejlene der opstår i forbindelse med databaseoperationer. Eksempelvis i UserMapper.SignUp() fanges en SQLException, og fejlen kastes videre med en brugerdefineret fejlbesked. Dette er nemlig med til at forhindrer, at programmet crasher ved databaseproblemer og giver i stedet brugeren en relevant fejlmeddelelse. Derudover logges fejl i HomeController, hvor lOGGER.severe (”Error during

login: " + e.getMessage()); her registrerer eventuelle problemer under loginprocessen. Det er med til at let gør fejlfinde og forbedre systemet.

Validering af brugerinput

Burgerinput valideres flere steder i vores program, for at sikre korrekt data og forhindre fejl. Før en ny bruger oprettes tjekkes der, om brugeren allerede findes i databasen via metoden UserMapper.userExists(user);. Ved login foretages der en SELECT-forespørgsler for at matche email og password med en eksisterende bruger. Hvis brugeren allerede er registreret, får brugeren en besked ctx.attribute("message", "Bruger findes allerede. Vær venlig at logge ind.");. Dette sikrer en bedre brugeroplevelse ved at give tydelig feedback.

Brugersikkerhed

En af vores sikkerhedsfunktioner under oprettelse af en konto, er at tjekke om brugeren allerede findes i databasen. Det gør vi via. en metode der sammenligner brugerens input af email med databasens indhold af e-mails.

En anden sikkerhed vi benytter os af i vores mappers er Prepared Statements. Den sørger for at man ikke kan lave SQL injections, hvilket betyder at vi kun kan modtage data og ikke nogle SQL forespørgsler fra brugerinput.

En væsentlig forbedring i brugersikkerheden kunne opnås ved at hash'e adgangskoder, inden de gemmes i databasen. I den nuværende løsning gemmes passwords i klartekst, hvilket udgør en sikkerhedsrisiko. Vi var lidt tidspressede, så det nåede vi ikke.

Ekstra elementer

En implementering vi blev enige om i vores kode er en dependency injection af vores ConnectionPool instans. Men i stedet for at lave en konstruktør med instansen, lavede vi i stedet en setter til ConnectionPool, i alle klasser der bruger ConnectionPool. Dette betyder at alle vores metoder skal være statiske. Udfra en setter kunne vi kalde på setter-metoden i main i stedet for at lave et objekt af hver klasse. Vi anvender dependency injection for at gøre vores kode mere

overskuelig og effektiv, da fordelen med dependency injection er, at man ikke behøver at gentage kodning i alle metoderne der benytter ConnectionPool.

Vi har struktureret vores kode i specifikke mapper/foldere og packages, f.eks den måde vi håndterer mvc på. Vi ved at vores controllers package kun indeholder de klasser der kommunikerer med vores html vha. Thymeleaf eller kommunikerer med vores mappers package. Dets formål er at behandle brugerens input og sende de håndterede data tilbage til brugeren. Det samme gælder vores mappers package, hvis primære formål er at kommunikere med databasen.

Status på implementation

Ved rapportens aflevering, er user story 1-6 blevet implementeret, dog ikke 7-9, og systemets udførelse stemmer overens med både aktivitets- og navigationsdiagrammerne. Til dette var det ikke nødvendigt at danne alle CRUD-metoderne for tabellerne, og der er derfor kun dannet de metoder, der var relevante til de formål, herunder primært SELECT- og INSERT-metoder, og én UPDATE-metode, til at ændre en brugers konto, når et køb udføres.

En funktion, der heller ikke blev implementeret, ville være en log-ud funktion. Denne var ikke beskrevet som værende en del af forventet funktionalitet, men givet at det er en åbenlys og praktisk funktion at besidde, når man arbejder med et loginsystem, er dens manglende tilstedeværelse værd at pointere. Ligesom user story 7-9, er denne ikke inkluderet grundet tidspres, og et fokus på en fuldendt implementation af de 6 første user stories, frem for et bredere udbud af funktioner.

Det er derudover også muligt, at en systematisk ændring, hvor sammensatte cupcakes kunne erstattes med orderdetails, for tættere tilknytning til databasen, samt færre overflødige klasser, kunne have været fordelagtig på lang sigt. Men da sådan ændring kunne have uforudsigelige konsekvenser for programmets evne til at udføre dets nuværende funktioner, var sådan ændringer ikke den optimale måde at bruge kostbar tid på.

Proces

Til at starte med gennemgik vi opgavebeskrivelsen, og prøvede på at fordele rollerne i gruppen ift. java-guru, scrum-master osv.. Vi kom frem til at Rasmus fik rollen som java-guru, da han har øje for detaljer og et meget godt overblik ift. koden. Alt andet ansvar tog vi sammen som gruppe. Vi lavede derefter en kort gruppekontrakt, som indholdt blandet andet at alle skulle blive hørt, og alle har sin forskellige mening, om hvordan koden eller hjemmesiden skal se ud. Vi aftalte, at vi primært sad på skolen 9:00-12:30/13:00, og vi kunne tage opgaver med hjem, hvis det gav mening.

Vi påbegyndte derefter at lave vores ERD diagram i fællesskab, som skulle være fundamentet for vores database. Efter den længe diskuterede databasestruktur, fordelte vi så småt opgaver ud til alle mand. Vi startede med at lave en mock up inde i figma, som skulle være det primære design for vores hjemmeside. Mens mock-uppen blev lavet, fik vi implementeret vores entiteter, og dele af vores cupcakemapper. Resten af den første uge blev brugt på at implementere vores mappers og controllers. Mandagen i 2. uge snakkede vi om hvad vi manglede og hvad vi havde nået hver især, vi havde overordnet set implementeret alle metoder i vores mappers og controllers. Vi begyndte derfor at knytte vores kode til html og thymeleaf. Vi fik brygget det sammen, sådan at vi fremviser bygning af brugerens email, cupcake, indkøbskurv, betaling af kurv, ordre og detaljer af ordren. Det kom selvfølgelig ikke uden et hav af problemer og udfordringer, som vi i sidste ende heldigvis fik løst.

Undervejs i projektet gik det op for os, at det egentlig er gået okay ift. kode og planlægning. Men vi ville gerne have nået mere, og stresse mindre så sent i forløbet. Alt i alt har der været forståelse i vores samarbejde, og alle meninger er blevet hørt, og hvis man kunne bidrage med noget, har man ikke været bange for at sige det højt. Kommunikationen hvor vi har arbejdet hjemme, og været på skolen har haft en god balance, hvor vi alle har været enige.

Alt i alt i sidste ende har vi en masse gode ting og erfaringer at tage med videre til fremtidige gruppeprojekter, men også noget vi skal blive bedre til som gruppe. Vi har snakket om, at vi vi skal blive bedre til følgende som gruppe:

- Vi skal sætte nogen strammere deadlines for os selv, så vi kan holde et godt flow og holde en god struktur i vores planlægning og dermed arbejdsproces. Vi skal på samme tid også presse os selv noget mere tidligere i projektet, selvom det virker surt, og laver lidt mindre tid til fritid. Ved at presse os selv tidligere og sætte stramme deadlines kan vi komme godt på forkant med projektet og hellere gerne være foran end at "holde" tidsplanen eller være bagud. Fordi det er svært at ligge en tidsplan, når man ikke ved hvad man render ind i af fejl, og hvordan det påvirker den resterende kode.

- Der skal laves diagrammer og en god struktur til at starte med, sådan at alle ved hvad der skal ske overordnet, og så kan der justeres undervejs hvis man finder noget der skal rettes eller mangler. Med diagrammer og et godt fundament for projektet, kan man se fremad, og hoppe videre, hvis man er blevet hurtigt færdig. Hvis man har tid, så hopper man videre, hvis det giver mening, sådan at vi kan beholde "forspringet", indtil vi støder ind i fejl, og kan bruge den indhentede tid på det, istedet for at sidde til sidst i projektet sidde og stresse over en masse kode og fejl, som bringer os bagud, og kan skabe frustration og gøre os ufokuserede.

- I og med at vi gerne vil have den gode struktur og prøve at skabe en god forkant. Så skal en del af planlægningen også være at få testet vores kode godt igennem, ved hjælp af unit-testing eller hvordan vi nu vil teste.

- Mens vi bliver færdige med diverse dele af koden, så i stedet for at vi laver hele backend først, så prøve at implementere det ind i html og thymeleaf sideløbende. Man skal selvfølgelig bruge forskellige dele af koden fra andre også, men man implementere hvis det giver mening og der er mulighed for det.

- Imens der bliver kodet, skal vi huske at få pushed og merget vores branches, så vi hele tiden holder vores master opdateret. Ellers som vi har oplevet her i cupcake projektet, at nogen sidder på en branch i to dage, mens at master bliver opdateret med en hel masse ny kode, så giver det en masse komplikationer.

video-demo:
https://www.youtube.com/watch?v=CYF2lWVHxSs